

# Two betting strategies that predict all compressible sequences

T. Petrović

tomeepx@gmail.com

Seventh International Conference on Computability, Complexity and Randomness (CCR 2012)

# Outline

## 1 Betting Games

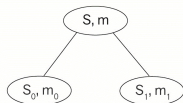
- Definition of Sequence-set Betting
- Comparison with Martingale Processes
- Comparison with Non-monotonic Betting

## 2 Algorithm that Constructs Two Betting Strategies

# Definition of Sequence-set Betting

(using mass-distribution)

- Start with  $(S, m)$
- Betting decision  $(S_0, m_0)$  ,  $(S_1, m_1)$   
 $m_0 + m_1 = m$   
 $\lambda(S_0) = \lambda(S_1) = \frac{1}{2}\lambda(S)$
- Sequence in  $S_0$ , next is  $(S_0, m_0)$   
 otherwise start with  $(S_1, m_1)$
- Success: capital  $c = m/\lambda(S)$   
 rises unboundedly
- Decision tree describes betting strategy



# Definition of Sequence-set Betting

(using mass-distribution)

- Start with  $(S, m)$
- Betting decision  $(S_0, m_0)$  ,  $(S_1, m_1)$   
 $m_0 + m_1 = m$   
 $\lambda(S_0) = \lambda(S_1) = \frac{1}{2}\lambda(S)$
- Sequence in  $S_0$ , next is  $(S_0, m_0)$   
 otherwise start with  $(S_1, m_1)$
- Success: capital  $c = m/\lambda(S)$   
 rises unboundedly
- Decision tree describes betting strategy



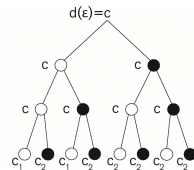
# Martingale Processes

- Function  $d$  from words to reals
- Equivalence relation on words  $\approx_d$   
 $v \approx_d w$  if for  
 $v' \preceq v, w' \preceq w, l(v') = l(w')$   
 we have  $d(v') = d(w')$

- Fairness condition:

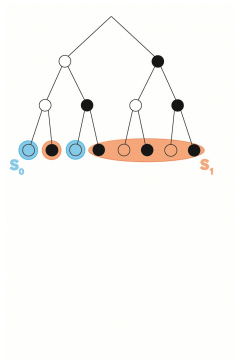
$$2 \sum_{\{v: v \approx_d w\}} d(v)$$

$$= \sum_{\{v: v \approx_d w\}} [d(v0) + d(v1)]$$



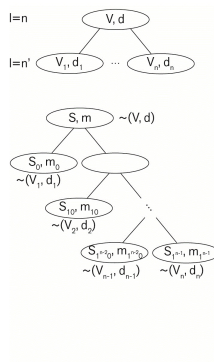
# Modified Sequence-set Betting

- Replace  $\lambda(S_0) = \lambda(S_1) = \frac{1}{2}\lambda(S)$  with  $\lambda(S_0) + \lambda(S_1) = \lambda(S)$
- Betting game equivalent to martingale processes



# Betting Strategy from Martingale Process

- $(S, m) \sim (V, d)$  ,  $V = \{v : v \approx_d w\}$   
 $S \prec \alpha \iff V \prec \alpha$  and  $m = d(w)\lambda(V)$
- Find  $v'$  s.t.  $v \prec v'$ ,  $d(v') \neq d(v)$
- $d$  divides  $V$  into  $V_1, \dots, V_n$
- Make according betting decisions
- No such  $v'$ , make no further bets



# Martingale Process from Betting Strategy

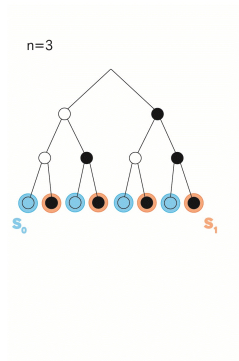
- $(S, m) \sim (V, d)$  ,  $V = \{v : v \approx_d w\}$   
 $S \prec \alpha \iff V \prec \alpha$  and  $m = d(w)\lambda(V)$
- Betting decision  $(S_0, m_0)$  ,  $(S_1, m_1)$
- Find  $n'$   
 $n' > l(w)$  ,  $n' \geq \max(l(w) : w \in S_0 \cup S_1)$
- Set  $d(v') = m_0/\lambda(S_0)$  for  $S_0 \preceq v'$   
 and  $d(v') = m_1/\lambda(S_1)$  for  $S_1 \preceq v'$   
 where  $l(v') = n'$
- No betting decision for  $(S, m)$   
 Set  $d(v') = d(w)$  for  $V \prec v'$





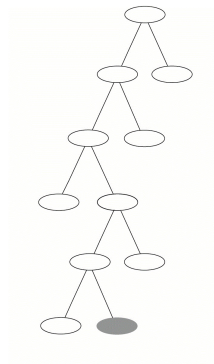
# Comparison with Non-monotonic Betting

- Add to req.  $\lambda(S_0) = \lambda(S_1) = \frac{1}{2}\lambda(S)$   
req. that for some  $n$   
 $w \in S_0 \Rightarrow w(n) = 0$   
 $w \in S_1 \Rightarrow w(n) = 1$
- Betting decision  $(S_0, m_0)$ ,  $(S_1, m_1)$   
places a bet on bit value at position  $n$
- Next iteration choses different position  
since previously picked positions have  
all 0's or all 1's



# Unpredictable Compressible Sequence

- Step1: For node  $(S, m)$  wait for a bet
- No betting decision, strategy fails
- Choose node with less mass
- If node small enough, compress all
- go to Step1



# Algorithm Outline

- Algorithm constructs betting decision trees for strategies  $A$  and  $B$
- The inputs are some word  $s$  and masses  $ma, mb$  assigned to that prefix by  $A$  and  $B$
- Calculates  $k$  from  $(s, ma, mb)$
- Runs UTM to enumerate first prefixes  $p$  that have inputs shorter by  $k$
- For each  $p$  adds betting decisions, nodes that contain  $p'$ ,  $p \prec p'$  contain only one word, same in  $A$  and  $B$ , for these start a new instance of algorithm
- If the set of prefixes is small, betting decisions such that either  $A$  or  $B$  double capital on that prefixes

# Notation

A  $a_1, \textcircled{S, ms+me} \neg L \dots a_x, \textcircled{S, ms+me} \neg L$     B  $b_1, \textcircled{S, ms+me} \neg L \dots b_y, \textcircled{S, ms+me} \neg L$

- Only the leaf nodes of b.d.t. that don't contain sequences on which another instance of algorithm was started are considered
- For each node  $(a_i, S, m)$ ,  $(b_j, S, m)$  two additional values are used
- Mass reserved to ensure that no node has mass 0,  $me$
- The portion of size of the so far found prefixes belonging to the node  $L$
- Mass  $ms$  is used for doubling the capital on compressible prefixes, mass assigned to node in b.d.t.  $m = ms + me$
- Denote indexes of considered leaf nodes of  $A$   $a_1, \dots, a_x$ , of  $B$   $b_1, \dots, b_y$
- $(a_i, S, ms, me, L)$   $S^{a_i}, ms^{a_i}, me^{a_i}, L^{a_i}$
- $(b_j, S, ms, me, L)$   $S_{b_j}, ms_{b_j}, me_{b_j}, L_{b_j}$

# Initialization

●  $(s, ma, mb)$

A  $a_i: (S, ms+me) - L$

B  $b_i: (S, ms+me) - L$

- The input for the algorithm instance is  $(s, ma, mb)$
- Set mass used for doubling the capital  $m = \min(ma, mb)/2$
- Set for A  $S^{a_1} = \{s\}$ ,  $ms^{a_1} = m$ ,  $me^{a_1} = ma - m$ ,  $L^{a_1} = 0$   
for B  $S_{b_1} = \{s\}$ ,  $ms_{b_1} = m$ ,  $me_{b_1} = mb - m$ ,  $L_{b_1} = 0$
- The capital for compressible prefixes will be  $c = 4(ma + mb)/\lambda(s)$
- set  $k$  such that  $2^{-k} < m^2(1 - cs)/2\lambda(s)c^2(1 + cs)^2$
- The  $cs$  is some constant  $0 < cs < 1$ , in all iterations  $S^{a_i} \cap S_{b_j} = \{\}$  or  $\lambda(s)\lambda(S^{a_i} \cap S_{b_j})$  is between  $(1 \pm cs)(\lambda(S^{a_i}) + L^{a_i})(\lambda(S_{b_j}) + L_{b_j})$
- The construction of b.d.t. starts by running the algorithm for  $(\varepsilon, 1, 1)$

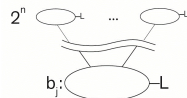
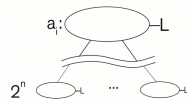
# Preparation Step

- Find next  $p$ ,  $S^{a_i} \cap S_{b_j} \cap p \neq \{\}$
- If for all  $a_i, b_j$   $S^{a_i} \cap S_{b_j} \cap p = \{\}$  or  $S^{a_i} \cap S_{b_j} \setminus p = \{\}$  skip rest
- For nodes  $a_i, b_j$  make  $n$  betting decisions,  $2^n$  new leaf nodes  $a_{ig}, b_{jh}$
- Evenly distribute mass and  $L$
- Extend words not in intersection by  $n$ , distribute them among leaf nodes
- Extend words in intersection and distribute them to have  
 $S^{a_{ig}} \cap S_{b_{jh}} \cap p = \{\}$  or  
 $S^{a_{ig}} \cap S_{b_{jh}} \setminus p = \{\}$

If  $\lambda(S^{a_i} \cap S_{b_j}) = r(\lambda(S^{a_i}) + L^{a_i})(\lambda(S_{b_j}) + L_{b_j})$  then  
 $\lambda(S^{a_{ig}} \cap S_{b_{jh}}) = r(\lambda(S^{a_{ig}}) + L^{a_{ig}})(\lambda(S_{b_{jh}}) + L_{b_{jh}})$

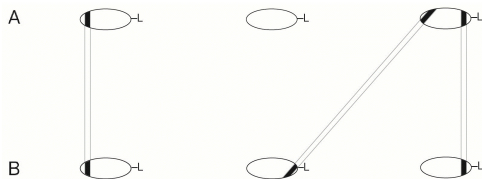
# Preparation Step

- Find next  $p$ ,  $S^{a_i} \cap S_{b_j} \cap p \neq \{\}$
- If for all  $a_i, b_j$   $S^{a_i} \cap S_{b_j} \cap p = \{\}$  or  $S^{a_i} \cap S_{b_j} \setminus p = \{\}$  skip rest
- For nodes  $a_i, b_j$  make  $n$  betting decisions,  $2^n$  new leaf nodes  $a_{ig}, b_{jh}$
- Evenly distribute mass and  $L$
- Extend words not in intersection by  $n$ , distribute them among leaf nodes
- Extend words in intersection and distribute them to have  
 $S^{a_{ig}} \cap S_{b_{jh}} \cap p = \{\}$  or  
 $S^{a_{ig}} \cap S_{b_{jh}} \setminus p = \{\}$



If  $\lambda(S^{a_i} \cap S_{b_j}) = r(\lambda(S^{a_i}) + L^{a_i})(\lambda(S_{b_j}) + L_{b_j})$  then  
 $\lambda(S^{a_{ig}} \cap S_{b_{jh}}) = r(\lambda(S^{a_{ig}}) + L^{a_{ig}})(\lambda(S_{b_{jh}}) + L_{b_{jh}})$

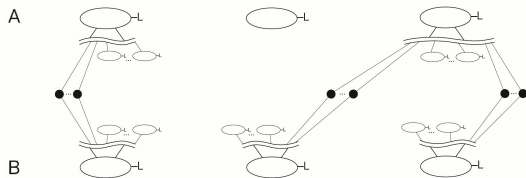
# Mass Assignment Step



- Pick  $S^{a_i} \cap S_{b_j}$ ,  $S^{a_i} \cap S_{b_j} \setminus p = \{\}$ , assign mass  $d^{a_i}$ ,  $d_{b_j}$  from  $ms^{a_i}$ ,  $ms_{b_j}$
- $d^{a_i} + d_{b_j} = c\lambda(S^{a_i} \cap S_{b_j})$
- If  $(ms^{a_i} - c\lambda(S^{a_i} \cap S_{b_j})) / (\lambda(S^{a_i}) + L^{a_i}) \geq ms_{b_j} / (\lambda(S_{b_j}) + L_{b_j})$  then  $d_{b_j} = 0$
- If  $(ms_{b_j} - c\lambda(S^{a_i} \cap S_{b_j})) / (\lambda(S_{b_j}) + L_{b_j}) \geq ms^{a_i} / (\lambda(S^{a_i}) + L^{a_i})$  then  $d^{a_i} = 0$
- otherwise find  $(ms^{a_i} - d^{a_i}) / (\lambda(S^{a_i}) + L^{a_i}) = (ms_{b_j} - d_{b_j}) / (\lambda(S_{b_j}) + L_{b_j})$



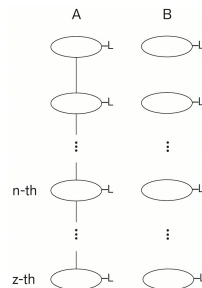
# Betting Decisions Step



- Add the size of all words of a node that are extensions of  $p$  to  $L$
- For nodes that contain words  $p'$  s.t.  $p \prec p'$  add betting decisions:
- Leaf nodes with extensions of  $p$  contain only one word, assign mass from previous step, start another instance of the algorithm
- Distribute words unrelated to  $p$  among remaining leaf nodes. Distribute  $L$  and the remainder of mass evenly among the remaining leaf nodes.
- For the remaining nodes we have  $S^{a_i} \cap S_{b_j} = \{\}$  or  $\lambda(s)\lambda(S^{a_i} \cap S_{b_j})$  between  $(1 \pm cs)(\lambda(S^{a_i}) + L^{a_i})(\lambda(S_{b_j}) + L_{b_j})$

# Proof Sketch

- $\Delta ms^{a_i} = ms^{a_i} - ms'^{a_i}$   
 $f = \frac{\lambda(s)}{m} ms^{a_i} / (\lambda(S^{a_i}) + L^{a_i})$   
 $f' = \frac{\lambda(s)}{m} ms'^{a_i} / (\lambda(S^{a_i}) + L^{a_i})$   
 $\Delta ms^{a_i} = \frac{m}{\lambda(s)} (f - f') (\lambda(S^{a_i}) + L^{a_i})$
- $\Delta ms^{a_i} \leq c \lambda(S^{a_i} \cap S_{b_j})$   
 $\lambda(S^{a_i} \cap S_{b_j})$  is close to  
 $\frac{1}{\lambda(s)} (\lambda(S^{a_i}) + L^{a_i}) (\lambda(S_{b_j}) + L_{b_j})$
- $g = \frac{\lambda(s)}{m} ms_{b_j} / (\lambda(S_{b_j}) + L_{b_j})$   
 $cL_{b_j} \geq (1 - g) \frac{m}{\lambda(s)} (\lambda(S_{b_j}) + L_{b_j})$   
 Assume that  $g \leq f'$
- $L_{b_j} \geq \text{const.} m^2 (1 - f') (f - f')$
- $\sum_{i=1}^z 2^{-l(p_i)} \geq \text{const.} m^2 / 2 \geq 2^{-k}$



# References



M. Li, P. Vitanyi

An Introduction to Kolmogorov Complexity and Its Applications,  
second edition

Springer, 1997.



J.M.Hitchcock, J.H. Lutz

Why Computational Complexity Requires Stricter Martingales

*Theory of Computing Systems*, 2006.



W. Merkle, N. Mihailović, T.A. Slaman

Some results on effective randomness

*Theory of Computing Systems*, 2006.